Lab 04

David Kirby

Due: 27 April 2020

1. **In our pipeline, can the data hazard below be resolved with forwarding alone? Why or why not?**

   ```
   lw $t0, 0($t1)
   sw $t0, 0($t2)
   ```

   The data hazard can be resolved with forwarding alone. This is because there is a forwarding MUX implemented in the execute stage that allows us to resolve the data hazard.

2. **Will the pipeline control unit slip the pipeline in the case of the instruction sequence above? Why or why not?**

   Yes, the pipeline control unit will slip the pipeline in the case of the instruction sequence above as it is designed to stall the fetch and decode stages, allowing the load to proceed down the pipeline.

Source Code

ForwardingUnit.vhd

```
1  ----------------------------------------------------
2  -- This component describes the forwarding unit in our
3  -- 5-stage, pipelined MIPS processor.
4  ----------------------------------------------------
5
6  library ieee;
7  use ieee.std_logic_1164.all;
8
9  ----------------------------------------------------
10 entity ForwardingUnit is
11
12     port (
13         UseShamt, UseImmed : in std_logic;
14         EX_RegWrEn, MEM_RegWrEn : in std_logic;
15         ID_Rs, ID_Rt : in std_logic_vector(4 downto 0);
16         EX_WrReg, MEM_WrReg : in std_logic_vector(4 downto 0);
17         AluSrcA, AluSrcB : out std_logic_vector(1 downto 0);
18         DataMemForwardCtrl_EX : out std_logic;
19         DataMemForwardCtrl_MEM : out std_logic
20     );
21 end ForwardingUnit;
22 ----------------------------------------------------
23
24 architecture behv of ForwardingUnit is
25
26 begin
27     ----------------------------------------------------
28     --forwarding logic for ALU operand A
29     --The sensitivity list! Do something if any of these signals change
30     process (UseShamt, ID_Rs, EX_WrReg, MEM_WrReg, EX_RegWrEn, MEM_RegWrEn)
31     begin --priority is important!
32         --If shift amount instruction, don't forward
33         if (UseShamt = '1') then
34             AluSrcA <= "00";--use shift amount
35             --else if data is produced by instruction one stage ahead
36         elsif ((EX_RegWrEn = '1') and (ID_Rs = EX_WrReg) and (EX_WrReg /= b"00000")) then
37             AluSrcA <= "10";--use EX bypass
38             --else if data is produced by instruction two stages ahead
39         elsif ((MEM_RegWrEn = '1') and (ID_Rs = MEM_WrReg) and (MEM_WrReg /= b"00000")) then
40             AluSrcA <= "01";--use MEM bypass
41         else
42             --else if data is produced by instruction in WB or beyond
43             AluSrcA <= "11";--pull from Register file
44         end if;
45     end process;
46
47     ----------------------------------------------------
48     --forwarding logic for ALU operand B
49     --The sensitivity list! Do something if any of these signals change
50     process (UseImmed, ID_Rt, EX_WrReg, MEM_WrReg, EX_RegWrEn, MEM_RegWrEn)
```

```
    begin --priority is important!
        --If immediate instruction, don't forward
        if (UseImmed = '1') then
            AluSrcB <= "00";--use immediate
            --else if data is produced by instruction one stage ahead
        elsif ((EX_RegWrEn = '1') and (ID_Rt = EX_WrReg) and (EX_WrReg /= b"00000")) then
            AluSrcB <= "10";--use EX bypass
            --else if data is produced by instruction two stages ahead
        elsif ((MEM_RegWrEn = '1') and (ID_Rt = MEM_WrReg) and (MEM_WrReg /= b"00000")) then
            AluSrcB <= "01";--use MEM bypass
        else
            --else if data is produced by instruction in WB or beyond
            AluSrcB <= "11";--pull from Register file
        end if;
    end process;

    ----------------------------------------------------
    --forwarding logic for data memory
    --The sensitivity list! Do something if any of these signals change
    process (ID_Rt, EX_WrReg, MEM_WrReg, EX_RegWrEn, MEM_RegWrEn)
    begin --priority is important!
        --if data is produced by instruction one stage ahead
        if ((EX_RegWrEn = '1') and (ID_Rt = EX_WrReg) and (EX_WrReg /= b"00000")) then
            DataMemForwardCtrl_EX <= '0';
            DataMemForwardCtrl_MEM <= '1';
            --else if data is produced by instruction two stages ahead
        elsif ((MEM_RegWrEn = '1') and (ID_Rt = MEM_WrReg) and (MEM_WrReg /= b"00000")) then
            DataMemForwardCtrl_EX <= '1';
            DataMemForwardCtrl_MEM <= '0';
            --else if data is produced by instruction in WB or beyond
        else
            DataMemForwardCtrl_EX <= '0';
            DataMemForwardCtrl_MEM <= '0';
        end if;
    end process;
    ----------------------------------------------------
end behv;
```

PipelineCtrl.vhd

```
----------------------------------------------------
-- This component describes the pipeline control unit
-- in our 5-stage, pipelined MIPS processor.

----------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity PipelineCtrl is

    port (
        EX_Branch : in std_logic;
        EX_Equals : in std_logic;
```

```
14          ID_Jump : in std_logic;
15          ID_Rs, ID_Rt : std_logic_vector(4 downto 0);
16          UseShamt, UseImmed : in std_logic;
17          EX_MemRdEn : in std_logic;--to detect a load
18          EX_WrReg : in std_logic_vector(4 downto 0);
19          PCwrite : out std_logic;
20          addrSel : out std_logic_vector(1 downto 0);
21          Flush_IF_ID, WrEn_IF_ID : out std_logic;
22          Flush_ID_EX : out std_logic
23      );
24  end PipelineCtrl;
25
26  architecture behv of PipelineCtrl is
27
28      begin
29      ----------------------------------------------------
30      --Pipeline control
31      --The sensitivity list! Do something if any of these signals change
32      process (UseShamt, ID_Rs, EX_WrReg, MEM_WrReg, EX_RegWrEn, MEM_RegWrEn)
33      begin --priority is important!
34          --If "Taken" branch instruction is detected in the execute stage
35          if (EX_Branch = '1') then
36              Flush_IF_ID <= "00";--Flush Fetch stage
37          elsif then
38          elsif then
39          else
40          end if;
41      end process;
```